Recurrent Neural Networks as Weighted Language Recognizers

Recurrent Neural Networks as Weighted Language Recognizers

Yining Chen
Dartmouth College
yining.chen.18@dartmouth.edu

Sorcha Gilroy
ILCC
University of Edinburgh
s.gilroy@sms.ed.ac.uk

Andreas Maletti
Institute of Computer Science
Universität Leipzig
andreas.maletti@uni-leipzig.de

Jonathan May
Information Sciences Institute
University of Southern California
jonmay@isi.edu

Kevin Knight
Information Sciences Institute
University of Southern California
knight@isi.edu

Abstract

We investigate the computational complexity of various problems for simple recurrent neural networks (RNNs) as formal models for recognizing weighted languages. We focus on the single-layer, ReLU-activation, rational-weight RNNs with softmax, which are commonly used in natural language processing applications. We show that most problems for such RNNs are undecidable, including consistency, equivalence, minimization, and the determination of the highest-weighted string. However, for consistent RNNs the last problem becomes decidable, although the solution length can surpass all computable bounds. If additionally the string is limited to polynomial length, the problem becomes NP-complete. In summary, this shows that approximations and heuristic algorithms are necessary in practical applications of those RNNs.

1 Introduction

Recurrent neural networks (RNNs) are an attractive apparatus for probabilistic language modeling (Mikolov and Zweig, 2012). Recent experiments show that RNNs significantly outperform other methods in assigning high probability to held-out English text (Jozefowicz et al., 2016).

Roughly speaking, an RNN works as follows. At each time step, it consumes one input token, updates its hidden state vector, and predicts the next token by generating a probability distribution over all permissible tokens. The probability of an input string is simply obtained as the product of the predictions of the tokens constituting the string followed by a terminating token. In this manner, each RNN defines a *weighted language*; i.e. a total function from strings to weights. Siegelmann and Sontag (1995) showed that single-layer rational-weight RNNs with saturated linear activation can compute any computable function. To this end, a specific architecture with 886 hidden units can simulate any Turing machine in real-time (i.e., each Turing machine step is simulated in a single time step). However, their RNN encodes the whole input in its internal state, performs the actual computation of the Turing machine when reading the terminating token, and then encodes the output (provided an output is produced) in a particular hidden unit. In this way, their RNN allows "thinking" time (equivalent to the computation time of the Turing machine) after the input has been encoded.

We consider a different variant of RNNs that is commonly used in natural language processing applications. It uses ReLU activations, consumes an input token at each time step, and produces softmax predictions for the next token. It thus immediately halts after reading the last input token and the weight assigned to the input is simply the product of the input token predictions in each step.

Other formal models that are currently used to implement probabilistic language models such as finite-state automata and context-free grammars are by now well-understood. A fair share of their utility directly derives from their nice algorithmic properties. For example, the weighted languages computed by weighted finite-state automata are closed under intersection (pointwise product) and union (pointwise sum), and the corresponding unweighted languages are closed under intersection, union, difference, and complementation (Droste et al., 2013). Moreover, toolkits like OpenFST (Allauzen et al., 2007) and Carmel[1] implement efficient algorithms on automata like minimization, intersection, finding the highest-weighted path and the highest-weighted string.

RNN practitioners naturally face many of these same problems. For example, an RNN-

[1] https://www.isi.edu/licensed-sw/carmel/

Proceedings of NAACL-HLT 2018, pages 2261–2271
New Orleans, Louisiana, June 1 - 6, 2018.

based machine translation system should extract the highest-weighted output string (i.e., the most likely translation) generated by an RNN, (Sutskever et al., 2014; Bahdanau et al., 2014). Currently this task is solved by approximation techniques like heuristic greedy and beam searches. To facilitate the deployment of large RNNs onto limited memory devices (like mobile phones) minimization techniques would be beneficial. Again currently only heuristic approaches like knowledge distillation (Kim and Rush, 2016) are available. Meanwhile, it is unclear whether we can determine if the computed weighted language is consistent; i.e., if it is a probability distribution on the set of all strings. Without a determination of the overall probability mass assigned to all finite strings, a fair comparison of language models with regard to perplexity is simply impossible.

The goal of this paper is to study the above problems for the mentioned ReLU-variant of RNNs. More specifically, we ask and answer the following questions:

- Consistency: Do RNNs compute consistent weighted languages? Is the consistency of the computed weighted language decidable?
- Highest-weighted string: Can we (efficiently) determine the highest-weighted string in a computed weighted language?
- Equivalence: Can we decide whether two given RNNs compute the same weighted language?
- Minimization: Can we minimize the number of neurons for a given RNN?

2 Definitions and notations

Before we introduce our RNN model formally, we recall some basic notions and notation. An *alphabet* Σ is a finite set of symbols, and we write $|\Sigma|$ for the number of symbols in Σ. A *string* s over the alphabet Σ is a finite sequence of zero or more symbols drawn from Σ, and we write Σ^* for the set of all strings over Σ, of which ϵ is the empty string. The length of the string $s \in \Sigma^*$ is denoted $|s|$ and coincides with the number of symbols constituting the string. As usual, we write A^B for the set of functions $\{f \mid f\colon B \to A\}$. A *weighted language* L is a total function $L\colon \Sigma^* \to \mathbb{R}$ from strings to real-valued weights. For example, $L(a^n) = e^{-n}$ for all $n \geq 0$ is such a weighted language.

We restrict the weights in our RNNs to the rational numbers $\mathbb{Q}$. In addition, we reserve the use of a special symbol \$ to mark the start and end of an input string. To this end, we assume that $\$ \notin \Sigma$ for all considered alphabets, and we let $\Sigma_\$ = \Sigma \cup \{\$\}$.

Definition 1. *A single-layer RNN R is a 7-tuple $\langle \Sigma, N, h_{-1}, W, W', E, E' \rangle$, in which*

- *Σ is an* input alphabet,
- *N is a finite set of* neurons,
- *$h_{-1} \in \mathbb{Q}^N$ is an* initial activation vector,
- *$W \in \mathbb{Q}^{N \times N}$ is a* transition matrix,
- *$W' = (W'_a)_{a \in \Sigma_\$}$ is a $\Sigma_\$$-indexed family of* bias vectors *$W'_a \in \mathbb{Q}^N$,*
- *$E \in \mathbb{Q}^{\Sigma_\$ \times N}$ is a* prediction matrix, *and*
- *$E' \in \mathbb{Q}^{\Sigma_\$}$ is a* prediction bias vector.

Next, let us define how such an RNN works. We first prepare our input encoding and the effect of our activation function. For an input string $s = s_1 s_2 \cdots s_n \in \Sigma^*$ with $s_1, \ldots, s_n \in \Sigma$, we encode this input as $\$s\$$ and thus assume that $s_0 = \$$ and $s_{n+1} = \$$. Our RNNs use ReLUs (Rectified Linear Units), so for every $v \in \mathbb{Q}^N$ we let $\sigma\langle v \rangle$ (the ReLU activation) be the vector $\sigma\langle v \rangle \in \mathbb{Q}^N$ such that

$$\sigma\langle v \rangle(n) = \max\big(0, v(n)\big) \qquad \text{for every } n \in N.$$

In other words, the ReLUs act like identities on nonnegative inputs, but clip negative inputs to 0. We use softmax-predictions, so for every vector $p \in \mathbb{Q}^{\Sigma_\$}$ and $a \in \Sigma_\$$ we let

$$\operatorname{softmax}\langle p \rangle(a) = \frac{e^{p(a)}}{\sum_{a' \in \Sigma_\$} e^{p(a')}} \ .$$

RNNs act in discrete time steps reading a single letter at each step. We now define the semantics of our RNNs.

Definition 2. *Let $R = \langle \Sigma, N, h_{-1}, W, W', E, E' \rangle$ be an RNN, s an input string of length n and $0 \leq t \leq n$ a time step. We define*

- *the* hidden state vector $h_{s,t} \in \mathbb{Q}^N$ *given by*

$$h_{s,t} = \sigma\langle W \cdot h_{s,t-1} + W'_{s_t} \rangle \ ,$$

  where $h_{s,-1} = h_{-1}$ and we use standard matrix product and point-wise vector addition,
- *the* next-token prediction vector $E_{s,t} \in \mathbb{Q}^{\Sigma_\$}$

$$E_{s,t} = E \cdot h_{s,t} + E'$$

- *the* next-token distribution $E'_{s,t} \in \mathbb{R}^{\Sigma_\$}$

$$E'_{s,t} = \operatorname{softmax}\langle E_{s,t} \rangle \ .$$

Finally, the RNN R computes the weighted language $R\colon \Sigma^* \to \mathbb{R}$, which is given for every input $s = s_1 \cdots s_n$ as above by

$$R(s) = \prod_{t=0}^{n} E'_{s,t}(s_{t+1}) \ .$$

In other words, each component $h_{s,t}(n)$ of the hidden state vector is the ReLU activation applied to a linear combination of all the components of the previous hidden state vector $h_{s,t-1}$ together with a summand W'_{s_t} that depends on the t-th input letter s_t. Thus, we often specify $h_{s,t}(n)$ as linear combination instead of specifying the matrix W and the vectors W'_a. The semantics is then obtained by predicting the letters $s_1, \ldots, s_n$ of the input s and the final terminator \$ and multiplying the probabilities of the individual predictions.

Let us illustrate these notions on an example. We consider the RNN $\langle \Sigma, N, h_{-1}, W, W', E, E' \rangle$ with $\gamma \in \mathbb{Q}$ and

- $\Sigma = \{a\}$ and $N = \{1, 2\}$,
- $h_{-1} = (-1, 0)^T$ and

$$W = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad W'_{\$} = W'_a = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- $E(\$, \cdot) = (M + 1, -(M + 1))$ and $E(a, \cdot) = (1, -1)$ and
- $E'(\$) = -M$ and $E'(a) = 0$.

In this case, we obtain the linear combinations

$$h_{s,t} = \sigma \left\langle \begin{matrix} h_{s,t-1}(1) + 1 \\ h_{s,t-1}(1) \end{matrix} \right\rangle$$

computing the next hidden state components. Given the initial activation, we thus obtain $h_{s,t} = \sigma\langle t, t-1 \rangle$. Using this information, we obtain

$$\begin{aligned} E_{s,t}(\$) &= (M+1) \cdot (t - \sigma\langle t-1 \rangle) - M \\ E_{s,t}(a) &= t - \sigma\langle t-1 \rangle \ . \end{aligned}$$

Consequently, we assign weight $\frac{e^{-M}}{1+e^{-M}}$ to input ε, weight $\frac{1}{1+e^{-M}} \cdot \frac{e^1}{e^1+e^1}$ to a, and, more generally, weight $\frac{1}{1+e^{-M}} \cdot \frac{1}{2^n}$ to a^n.

Clearly the weight assigned by an RNN is always in the interval $(0, 1)$, which enables a probabilistic view. Similar to weighted finite-state automata or weighted context-free grammars, each RNN is a compact, finite representation of a weighted language. The softmax-operation enforces that the probability 0 is impossible as assigned weight, so each input string is principally possible. In practical language modeling, smoothing methods are used to change distributions such that impossibility (probability 0) is removed. Our RNNs avoid impossibility outright, so this can be considered a feature instead of a disadvantage.

The hidden state $h_{s,t}$ of an RNN can be used as scratch space for computation. For example, with a single neuron n we can count symbols in s via:

$$h_{s,t}(n) = \sigma\langle h_{s,t-1}(n) + 1 \rangle \ .$$

Here the letter-dependent summand W'_a is universally 1. Similarly, for an alphabet $\Sigma = \{a_1, \ldots, a_m\}$ we can use the method of Siegelmann and Sontag (1995) to encode the complete input string s in base $m + 1$ using:

$$h_{s,t}(n) = \sigma\langle (m+1) h_{s,t-1}(n) + c(s_t) \rangle \ ,$$

where $c\colon \Sigma_{\$} \to \{0, \ldots, m\}$ is a bijection. In principle, we can thus store the entire input string (of unbounded length) in the hidden state value $h_{s,t}(n)$, but our RNN model outputs weights at each step and terminates immediately once the final delimiter \$ is read. It must assign a probability to a string *incrementally* using the chain rule decomposition $p(s_1 \cdots s_n) = p(s_1) \cdot \ldots \cdot p(s_n \mid s_1 \cdots s_{n-1})$.

Let us illustrate our notion of RNNs on some additional examples. They all use the alphabet $\Sigma = \{a\}$ and are illustrated and formally specified in Figure 1. The first column shows an RNN R_1 that assigns $R_1(a^n) = 2^{-(n+1)}$. The next-token prediction matrix ensures equal values for a and \$ at every time step. The second column shows the RNN R_2, which we already discussed. In the beginning, it heavily biases the next symbol prediction towards a, but counters it starting at $t = 1$. The third RNN R_3 uses another counting mechanism with $h_{s,t} = \sigma\langle t - 100, t - 101, t \rangle$. The first two components are ReLU-thresholded to zero until $t > 101$, at which point they overwhelm the bias towards a turning all future predictions to \$.

3 Consistency

We first investigate the consistency problem for an RNN R, which asks whether the recognized weighted language R is indeed a probability distribution. Consequently, an RNN R is *consistent*

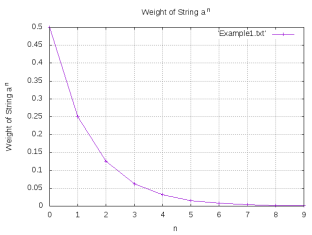
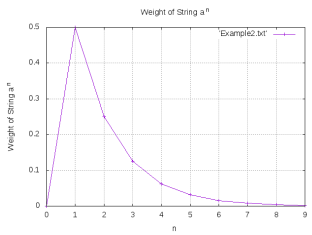
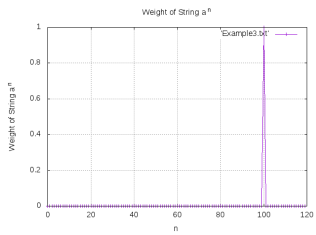
	$R_1(a^n) = 2^{-(n+1)}$	$R_2(\varepsilon) \approx 0$ $R_2(a^n) \approx 2^{-n}$ $(n \geq 1)$	$R_3(a^{100}) \approx 1$ $R_3(a^n) \approx 0$ $(n \neq 100)$
N	$\{1\}$	$\{1, 2\}$	$\{1, 2, 3\}$
h_{-1}	$\begin{pmatrix} 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
W	$\begin{pmatrix} 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$
$W'_{\$}$ W'_a	$\begin{pmatrix} 0 \end{pmatrix}$ $\begin{pmatrix} 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} -99 \\ -100 \\ 1 \end{pmatrix}$ $\begin{pmatrix} -99 \\ -100 \\ 1 \end{pmatrix}$
$E_{\$}$ E_a	$\begin{pmatrix} 0 \end{pmatrix}$ $\begin{pmatrix} 0 \end{pmatrix}$	$\begin{pmatrix} M+1 \\ -(M+1) \end{pmatrix}$ $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$	$\begin{pmatrix} M \\ -M \\ 0 \end{pmatrix}$ $\begin{pmatrix} -M \\ M \\ 0 \end{pmatrix}$
$E'_{\$}$ E'_a	0 0	$-M$ 0	$-M$ 0
			

Figure 1: Sample RNNs over single-letter alphabets, and the weighted languages they recognize. M is some positive rational number which depends on the desired error margin. If we want to express the second and the third languages with error margin δ, M is chosen so that $M > -\ln \frac{\delta}{1-\delta}$ in column 2, and chosen so that $(1 + e^{-M})^{100} < \frac{1}{1-\delta}$ in column 3.

if $\sum_{s \in \Sigma^*} R(s) = 1$. We first show that there is an inconsistent RNN, which together with our examples shows that consistency is a nontrivial property of RNNs.[2]

We immediately use a slightly more complex example, which we will later reuse.

Example 3. Let us consider an arbitrary RNN

$$R = \langle \Sigma, N, h_{-1}, W, W', E, E' \rangle$$

with the single-letter alphabet $\Sigma = \{a\}$, the neurons $\{1, 2, 3, n, n'\} \subseteq N$, initial activation $h_{-1}(i) = 0$ for all $i \in \{1, 2, 3, n, n'\}$, and the following linear combinations:

$$h_{s,t}(1) = \sigma\langle h_{s,t-1}(1) + h_{s,t-1}(n) - h_{s,t-1}(n') \rangle$$
$$h_{s,t}(2) = \sigma\langle h_{s,t-1}(2) + 1 \rangle$$
$$h_{s,t}(3) = \sigma\langle h_{s,t-1}(3) + 3h_{s,t-1}(1) \rangle$$
$$E_{s,t}(\$) = h_{s,t}(3) - h_{s,t}(2)$$
$$E_{s,t}(a) = h_{s,t}(2)$$

Now we distinguish two cases:
Case 1: If $h_{s,t}(n) - h_{s,t}(n') = 0$ for all $t \in \mathbb{N}$, then $h_{s,t}(1) = 0$ and $h_{s,t}(2) = t + 1$ and $h_{s,t}(3) = 0$. Hence we have $E_{s,t}(\$) = -(t + 1)$ and $E_{s,t}(a) = t + 1$. In this case the termination probability

$$E'_{s,t}(\$) = \frac{e^{-(t+1)}}{e^{-(t+1)} + e^{t+1}} = \frac{1}{1 + e^{2(t+1)}}$$

(i.e., the likelihood of predicting \$) shrinks rapidly towards 0, so the RNN assigns less than 15% of the probability mass to the terminating sequences (i.e., the finite strings), so the RNN is inconsistent (see Lemma 15 in the appendix).

Case 2: Suppose that there exists a time

[2] For comparison, all probabilistic finite-state automata are consistent, provided no transitions exit final states. Not all probabilistic context-free grammars are consistent; necessary and sufficient conditions for consistency are given by Booth and Thompson (1973). However, probabilistic context-free grammars obtained by training on a finite corpus using popular methods (such as expectation-maximization) are guaranteed to be consistent (Nederhof and Satta, 2006).

point $T \in \mathbb{N}$ such that for all $t \in \mathbb{N}$

$$h_{s,t}(n) - h_{s,t}(n') = \begin{cases} 1 & \text{if } t = T \\ 0 & \text{otherwise.} \end{cases}$$

Then $h_{s,t}(1) = 0$ for all $t \leq T$ and $h_{s,t}(1) = 1$ otherwise. In addition, we have $h_{s,t}(2) = t + 1$ and $h_{s,t}(3) = \sigma\langle 3(t - T - 1)\rangle$. Hence we have

$$\begin{aligned} E_{s,t}(\$) &= \sigma\langle 3(t - T - 1)\rangle - (t + 1) \\ &= \begin{cases} -(t+1) & \text{if } t \leq T \\ 2t - 3T - 4 & \text{otherwise} \end{cases} \\ E_{s,t}(a) &= t + 1 \ , \end{aligned}$$

which shows that the probability

$$E'_{s,t}(\$) = \begin{cases} \frac{1}{1+e^{2(t+1)}} & \text{if } t \leq T \\ \frac{e^{t-3T-5}}{1+e^{t-3T-5}} & \text{otherwise} \end{cases}$$

of predicting \$ increases over time and eventually (for $t \gg 3T$) far outweighs the probability of predicting a. Consequently, in this case the RNN is consistent (see Lemma 16 in the appendix).

We have seen in the previous example that consistency is not trivial for RNNs, which takes us to the consistency problem for RNNs:

Consistency: Given an RNN R, return "yes" if R is consistent and "no" otherwise.

We recall the following theorem, which, combined with our example, will prove that consistency is unfortunately undecidable for RNNs.

Theorem 4 (Theorem 2 of Siegelmann and Sontag (1995))**.** *Let M be an arbitrary deterministic Turing machine. There exists an RNN*

$$R = \langle \Sigma, N, h_{-1}, W, W', E, E' \rangle$$

with saturated linear activation, input alphabet $\Sigma = \{a\}$, and 1 designated neuron $n \in N$ such that for all $s \in \Sigma^*$ and $0 \leq t \leq |s|$

- *$h_{s,t}(n) = 0$ if M does not halt on ε, and*
- *if M does halt on empty input after T steps, then*

$$h_{s,t}(n) = \begin{cases} 1 & \text{if } t = T \\ 0 & \text{otherwise.} \end{cases}$$

In other words, such RNNs with saturated linear activation can semi-decide halting of an arbitrary Turing machine in the sense that a particular neuron achieves value 1 at some point during the evolution if and only if the Turing machine halts on empty input. An RNN with saturated linear activation is an RNN following our definition with the only difference that instead of our ReLU-activation σ the following saturated linear activation $\sigma' \colon \mathbb{Q}^N \to \mathbb{Q}^N$ is used. For every vector $v \in \mathbb{Q}^N$ and $n \in N$, let

$$\sigma'\langle v \rangle(n) = \begin{cases} 0 & \text{if } v(n) < 0 \\ v(n) & \text{if } 0 \leq v(n) \leq 1 \\ 1 & \text{if } v(n) > 1 \ . \end{cases}$$

Since $\sigma'\langle v \rangle = \sigma\langle v \rangle - \sigma\langle v - \vec{1} \rangle$ for all $v \in \mathbb{Q}^N$, and the right-hand side is a linear transformation, we can easily simulate saturated linear activation in our RNNs. To this end, each neuron $n \in N$ of the original RNN $R = \langle \Sigma, N, h_{-1}, U, U', E, E' \rangle$ is replaced by two neurons n_1 and n_2 in the new RNN $R' = \langle \Sigma, N', h'_{-1}, V, V', F, F' \rangle$ such that $h_{s,t}(n) = h'_{s,t}(n_1) - h'_{s,t}(n_2)$ for all $s \in \Sigma^*$ and $0 \leq t \leq |s|$, where the evaluation of $h'_{s,t}$ is performed in the RNN R'. More precisely, we use the transition matrix V and bias function V', which is given by

$$\begin{aligned} V(n_1, n'_1) &= V(n_2, n'_1) = U(n, n') \\ V(n_1, n'_2) &= V(n_2, n'_2) = -U(n, n') \\ V'_a(n_1) &= U'_a(n) \\ V'_a(n_2) &= U'_a(n) - 1 \\ h'_{-1}(n_1) &= h_{-1}(n) \\ h'_{-1}(n_2) &= 0 \end{aligned}$$

for all $n, n' \in N$ and $a \in \Sigma \cup \{\$\}$, where n_1 and n_2 are the two neurons corresponding to n and n'_1 and n'_2 are the two neurons corresponding to n' (see Lemma 17 in the appendix).

Corollary 5. *Let M be an arbitrary deterministic Turing machine. There exists an RNN*

$$R = \langle \Sigma, N, h_{-1}, W, W', E, E' \rangle$$

with input alphabet $\Sigma = \{a\}$ and 2 designated neurons $n_1, n_2 \in N$ such that for all $s \in \Sigma^*$ and $0 \leq t \leq |s|$

- *$h_{s,t}(n_1) - h_{s,t}(n_2) = 0$ if M does not halt on ε, and*
- *if M does halt on empty input after T steps, then*

$$h_{s,t}(n_1) - h_{s,t}(n_2) = \begin{cases} 1 & \text{if } t = T \\ 0 & \text{otherwise.} \end{cases}$$

We can now use this corollary together with the RNN R of Example 3 to show that the consistency problem is undecidable. To this end, we simulate a given Turing machine M and identify the two designated neurons of Corollary 5 as n and n' in Example 3. It follows that M halts if and only if R is consistent. Hence we reduced the undecidable halting problem to the consistency problem, which shows the undecidability of the consistency problem.

Theorem 6. *The consistency problem for RNNs is undecidable.*

As mentioned in Footnote 2, probabilistic context-free grammars obtained after training on a finite corpus using the most popular methods are guaranteed to be consistent. At least for 2-layer RNNs this does not hold.

Theorem 7. *A two-layer RNN trained to a local optimum using Back-propagation-through-time (BPTT) on a finite corpus is not necessarily consistent.*

Proof. The first layer of the RNN R with a single alphabet symbol a uses one neuron n' and has the following behavior:

$$h_{-1}(n') = 0$$
$$h_{s,t}(n') = \sigma\langle h_{s,t-1}(n') + 1\rangle$$

The second layer uses neuron n and takes $h_{s,t}(n')$ as input at time t:

$$h_{s,t}(n) = \sigma\langle h_{s,t}(n') - 2\rangle$$
$$E_{s,t}(a) = h_{s,t}(n) \qquad E_{s,t}(\$) = 0$$
$$E'_{s,t}(a) = \begin{cases} \frac{1}{2} & \text{if } t \leq 1 \\ \frac{e^{(t-1)}}{1+e^{(t-1)}} & \text{otherwise.} \end{cases}$$

Let the training data be $\{a\}$. Then the objective we wish to maximize is simply $R(a)$. The derivative of this objective with respect to each parameter is 0, so applying gradient descent updates does not change any of the parameters and we have converged to an inconsistent RNN. □

It remains an open question whether there is a single-layer RNN that also exhibits this behavior.

4 Highest-weighted string

Given a function $f\colon \Sigma^* \to \mathbb{R}$ we are often interested in the highest-weighted string. This corresponds to the most likely sentence in a language model or the most likely translation for a decoder RNN in machine translation.

	Best-path	Best-string
General RNN	**Undecidable**	
Consistent RNN	**NP-c** [3]	
Det. PFSA/PCFG	P [4]	
Nondet. PFSA/PCFG		NP-c [5]

Table 1: Comparison of the difficulty of identifying the most probable derivation (Best-path) and the highest-weighted string (Best-string) for various models.

For deterministic probabilistic finite-state automata or context-free grammars only one path or derivation exists for any given string, so the identification of the highest-weighted string is the same task as the identification of the most probable path or derivation. However, for nondeterministic devices, the highest-weighted string is often harder to identify, since the weight of a string is the sum of the probabilities of all possible paths or derivations for that string. A comparison of the difficulty of identifying the most probable derivation and the highest-weighted string for various models is summarized in Table 1, in which we marked our results in bold face.

We present various results concerning the difficulty of identifying the highest-weighted string in a weighted language computed by an RNN. We also summarize some available algorithms. We start with the formal presentation of the three studied problems.

1. **Best string:** Given an RNN R and $c \in (0, 1)$, does there exist $s \in \Sigma^*$ with $R(s) > c$?
2. **Consistent best string:** Given a consistent RNN R and $c \in (0, 1)$, does there exist $s \in \Sigma^*$ with $R(s) > c$?
3. **Consistent best string of polynomial length:** Given a consistent RNN R, polynomial $\mathcal{P}$ with $\mathcal{P}(x) \geq x$ for $x \in \mathbb{N}^+$, and $c \in (0, 1)$, does there exist $s \in \Sigma^*$ with $|s| \leq \mathcal{P}(|R|)$ and $R(s) > c$?

As usual the corresponding optimization problems are not significantly simpler than these decision problems. Unfortunately, the general problem is also undecidable, which can easily be shown using our example.

[3] Restricted to solutions of polynomial length
[4] Dijkstra shortest path / (Knuth, 1977)
[5] (Casacuberta and de la Higuera, 2000) / (Simaan, 1996)

Theorem 8. *The best string problem for RNNs is undecidable.*

Proof. Let M be an arbitrary Turing machine and again consider the RNN R of Example 3 with the neurons n and n' identified with the designated neurons of Corollary 5. We note that $R(\varepsilon) = \frac{1}{1+e^2} < 0.12$ in both cases. If M does not halt, then $R(a^n) \leq \frac{1}{1+e^{2(n+1)}} \leq \frac{1}{1+e^2} < 0.12$ for all $n \in \mathbb{N}$. On the other hand, if M halts after T steps, then

$$\begin{aligned} & R(a^{3T-5}) \\ &= \Big(\prod_{t=0}^{T} \frac{e^{2(t+1)}}{1+e^{2(t+1)}}\Big) \cdot \Big(\prod_{t=T+1}^{3T-6} \frac{1}{1+e^{t-3T-5}}\Big) \cdot \frac{1}{2} \\ &\geq \frac{2}{(-1,e^{-2})_\infty} \cdot \Big(\prod_{t=T+1}^{3T-6} \frac{e^{3T+5-t}}{e^{3T+5-t+1}}\Big) \cdot \frac{1}{2} \\ &\geq \frac{2}{(-1,e^{-2})_\infty \cdot (-1,e^{-1})_\infty} \geq 0.25 \end{aligned}$$

using Lemma 14 in the appendix. Consequently, a string with weight above 0.12 exists if and only if M halts, so the best string problem is also undecidable. □

If we restrict the RNNs to be consistent, then we can easily decide the best string problem by simple enumeration.

Theorem 9. *The consistent best string problem for RNNs is decidable.*

Proof. Let R be the RNN over alphabet Σ and $c \in (0,1)$ be the bound. Since Σ^* is countable, we can enumerate it via $f\colon \mathbb{N} \to \Sigma^*$. In the algorithm we compute $S_n = \sum_{i=0}^{n} R(f(i))$ for increasing values of n. If we encounter a weight $R(f(n)) > c$, then we stop with answer "yes." Otherwise we continue until $S_n > 1 - c$, at which point we stop with answer "no."

Since R is consistent, $\lim_{i\to\infty} S_i = 1$, so this algorithm is guaranteed to terminate and it obviously decides the problem. □

Next, we investigate the length $|w_R^{\max}|$ of the shortest string $w_R^{\max}$ of maximal weight in the weighted language R generated by a consistent RNN R in terms of its (binary storage) size $|R|$. As already mentioned by Siegelmann and Sontag (1995) and evidenced here, only small precision rational numbers are needed in our constructions, so we assume that $|R| \leq c \cdot |N|^2$ for a (reasonably small) constant c, where N is the set of neurons of R. We show that no computable bound on the length of the best string can exist, so its length can surpass all reasonable bounds.

Theorem 10. *Let $f\colon \mathbb{N}_+ \to \mathbb{N}$ be the function with*

$$f(n) = \max_{\substack{\text{consistent RNN } R \\ |R| \leq n}} |w_R^{max}|$$

for all $n \in \mathbb{N}_+$. There exists no computable function $g\colon \mathbb{N} \to \mathbb{N}$ with $g(n) \geq f(n)$ for all $n \in \mathbb{N}$.

Proof. In the previous section (before Theorem 6) we presented an RNN R_M that simulates an arbitrary (single-track) Turing machine M with n states. By Siegelmann and Sontag (1995) we have $|R_M| \leq c \cdot (4n + 16)$. Moreover, we observed that this RNN R_M is consistent if and only if the Turing machine M halts on empty input. In the proof of Theorem 8 we have additionally seen that the length $|w_R^{\max}|$ of its best string exceeds the number T_M of steps required to halt.

For every $n \in \mathbb{N}$, let $BB(n)$ be the n-th "Busy Beaver" number (Radó, 1962), which is

$$BB(n) = \max_{\substack{\text{normalized } n\text{-state Turing machine } M \text{ with} \\ \text{2 tape symbols that halts on empty input}}} T_M$$

It is well-known that $BB\colon \mathbb{N}_+ \to \mathbb{N}$ cannot be bounded by any computable function. However,

$$\begin{aligned} BB(n) &\leq \max_{\substack{\text{normalized } n\text{-state Turing machine } M \text{ with} \\ \text{and 2 tape symbols that halts on empty input}}} |w_{R_M}^{\max}| \\ &\leq \max_{\substack{\text{consistent RNN } R \\ |R| \leq c\cdot(4n+16)}} |w_R^{\max}| \\ &= f(4nc + 16c) \ , \end{aligned}$$

so f clearly cannot be computable and no computable function g can provide bounds for f. □

Finally, we investigate the difficulty of the best string problem for consistent RNN restricted to solutions of polynomial length.

Theorem 11. *Identifying the best string of polynomial length in a consistent RNN is NP-complete.*

Proof. To show NP-hardness, we reduce from the 3-SAT problem. Let $x_1, \ldots, x_m$ be m Boolean variables and

$$F = \bigwedge_{i=1}^{k} \Big(\ell_{i1} \vee \ell_{i2} \vee \ell_{i3}\Big) \ ,$$

be a formula in conjunctive normal form, where $\ell_{ij} \in \{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m\}$. 3-SAT asks whether there is a setting of x_is that makes F true.

We initialize $h_{-1}(n) = 0, \forall\ n \in N = \{x_1, \ldots, x_m, c_1, \ldots, c_k, c'_1, \ldots, c'_k, F, n_1, n_2, n_3, \star\}$. Let $s \in \{0,1\}^*$ be the input string. Denote the value of F when $x_j = s_j$ for all $j \in [m]$ as $F(s)$. Let $t \in \mathbb{N}$ with $t \leq |s|$. Set $h_{s,t}(x_m) = \sigma\langle I(s_t)\rangle$, where $I(0) = I(\$) = 0$ and $I(1) = 1$. This stores the current input symbol in neuron x_m, so $h_{s,t}(x_m) = I(s_t)$. In addition, we let $h_{s,t}(x_j) = \sigma\langle h_{s,t-1}(x_{j+1})\rangle$ for all $j \in [m-1]$. Consequently, for all $j \in [m]$

$$h_{s,t}(x_j) = \begin{cases} I(s_{t-(m-j)}) & \text{if } m-j \leq t \\ 0 & \text{otherwise.} \end{cases}$$

Next, we evaluate the clauses. For each $i \in [k]$, we use two neurons c_i and c'_i such that

$$h_{s,t}(c_i) = \sigma\langle f_{s,t}(\ell_{i1}) + f_{s,t}(\ell_{i2}) + f_{s,t}(\ell_{i3})\rangle$$
$$h_{s,t}(c'_i) = \sigma\langle f_{s,t}(\ell_{i1}) + f_{s,t}(\ell_{i2}) + f_{s,t}(\ell_{i3}) - 1\rangle,$$

where $f_{s,t}(x_m) = I(s_t)$, $f_{s,t}(\neg x_m) = 1 - I(s_t)$, and $\forall j \in [m-1], f_{s,t}(x_j) = h_{s,t-1}(x_{j+1})$, $f_{s,t}(\neg x_j) = 1 - h_{s,t-1}(x_{j+1})$. Note that $h_{s,t}(c_i) - h_{s,t}(c'_i)$ contains the evaluation of the clause $\ell_{i1} \vee \ell_{i2} \vee \ell_{i3}$. Let

$$h_{s,t}(F) = \sigma\Big\langle \sum_{i=1}^{k} \Big(h_{s,t-1}(c_i) - h_{s,t-1}(c'_i)\Big) - k + 1 \Big\rangle,$$

so $h_{s,t}(F) = F(s)$ contains the evaluation of the formula F using the values in neurons $x_1, \ldots, x_m$.

We use three counters n_1, n_2, n_3 to ensure that the only relevant inputs are of length $m+2$:

$$h_{s,t}(n_1) = \sigma\langle h_{s,t-1}(n_3) - (m+2)\rangle$$
$$h_{s,t}(n_2) = \sigma\langle h_{s,t-1}(n_3) - (m+1)\rangle$$
$$h_{s,t}(n_3) = \sigma\langle h_{s,t-1}(n_3) + 1\rangle\ ,$$

which yields $h_{s,t}(n_3) = t + 1$, $h_{s,t}(n_2) = \sigma\langle t - (m+1)\rangle$, and $h_{s,t}(n_1) = \sigma\langle t - (m+2)\rangle$.

Our goal neuron is $\star$, which we set to

$$h_{s,t}(\star) = \sigma\langle h_{s,t-1}(F) - h_{s,t-1}(n_1) + h_{s,t-1}(n_2) - 1\rangle$$

so that

$$h_{s,t}(\star) = \begin{cases} h_{s,t-1}(F) & \text{if } t = m+2 \\ 0 & \text{otherwise,} \end{cases}$$

so $h_{s,t}(\star) = 1$ if and only if $t = m+2$ and $F(s) = 1$.

Let $m' = m + 4$. The output is set as follows:

$$E_{s,t}(0) = E_{s,t}(1) = m'\big(1 - 2h_{s,t}(\star)\big)$$
$$E_{s,t}(\$) = -m'\big(1 - 2h_{s,t}(\star)\big)\ ,$$

This yields $E_{s,t}(0) = E_{s,t}(1) = -E_{s,t}(\$) = -m'$ if $t = m+2$ and $F(s) = 1$, and m' otherwise. For $a \in \{0,1\}$,

$$E'_{s,t}(a) = \begin{cases} \frac{e^{-m'}}{2e^{-m'}+e^{m'}} & \text{if } t = m+2 \text{ and } F(s) = 1 \\ \frac{e^{m'}}{2e^{m'}+e^{-m'}} & \text{otherwise} \end{cases}$$

$$E'_{s,t}(\$) = \begin{cases} \frac{e^{m'}}{2e^{-m'}+e^{m'}} & \text{if } t = m+2 \text{ and } F(s) = 1 \\ \frac{e^{-m'}}{2e^{m'}+e^{-m'}} & \text{otherwise.} \end{cases}$$

Finally, we set the threshold $\xi = 3^{-m'}$. When $|s| \neq m+2$, $s_{m+3} \neq \$$, so the weight of s contains the factor $\frac{e^{-m'}}{2e^{-m'}+e^{m'}} = \frac{1}{2+e^{2m'}}$ and thus is upper-bounded by $\frac{1}{2+e^{2m'}} < \xi$. Hence no input of length different from $m+2$ achieves a weight that exceeds ξ. A string s of length $m+2$ achieves the weight w_s given by

$$w_s = \begin{cases} \frac{e^{m'}}{2e^{-m'}+e^{m'}} \cdot \prod_{i=1}^{m+2} \frac{e^{m'}}{2e^{m'}+e^{-m'}} & \text{if } F(s) = 1 \\ \frac{e^{-m'}}{2e^{m'}+e^{-m'}} \cdot \prod_{i=1}^{m+2} \frac{e^{m'}}{2e^{m'}+e^{-m'}} & \text{otherwise.} \end{cases}$$

When $F(s) = 0$, $w_s < \frac{e^{-m'}}{2e^{m'}+e^{-m'}} < \xi$, so if F is unsatisfiable, no input string achieves a weight above the threshold ξ. When $F(s) = 1$, $w_s = \frac{e^{m'}}{2e^{-m'}+e^{m'}} \cdot \left(\frac{e^{m'}}{2e^{m'}+e^{-m'}}\right)^{m+2} > \xi$. An input string with weight above ξ exists if and only if F is satisfiable. Obviously, the reduction can be computed in polynomial time since all constants can be computed in logarithmic space. The constructed RNN is consistent, since the output prediction is constant after $m+3$ steps. □

5 Equivalence

We prove that equivalence of two RNNs is undecidable. For comparison, equivalence of two deterministic WFSAs can be tested in time $O(|\Sigma|(|Q_A| + |Q_B|)^3)$, where $|Q_A|$, $|Q_B|$ are the number of states of the two WFSAs and $|\Sigma|$ is the size of the alphabet (Cortes et al., 2007); equivalence of nondeterministic WFSAs are undecidable (Griffiths, 1968). The decidability of language equivalence for deterministic probabilistic pushdowntown automata (PPDA) is still open (Forejt et al., 2014), although equivalence for deterministic unweighted push-downtown automata (PDA) is decidable (Sénizergues, 1997).

The equivalence problem is formulated as follows:

Equivalence: Given two RNNs R and R', return "yes" if $R(s) = R'(s)$ for all $s \in \Sigma^*$, and "no" otherwise.

Theorem 12. *The equivalence problem for RNNs is undecidable.*

Proof. We prove by contradiction. Suppose Turing machine M decides the equivalence problem. Given any deterministic Turing Machine M', construct the RNN R that simulates M' on input ϵ as described in Corollary 5. Let $E_{s,t}(a) = 0$ and $E_{s,t}(\$) = h_{s,t}(n_1) - h_{s,t}(n_2)$. If M' does not halt on ϵ, for all $t \in \mathbb{N}$, $E'_{s,t}(a) = E'_{s,t}(\$) = 1/2$; if M' halts after T steps, $E'_{s,T}(a) = 1/(e+1)$, $E_{s,T}(\$) = e/(e+1)$. Let R' be the trivial RNN that computes $\{a^n : P(a^n) = 2^{-(n+1)}, n \geq 0\}$. We run M on input $\langle R, R' \rangle$. If M returns "no", M' halts on x, else it does not halt. Therefore the Halting Problem would be decidable if equivalence is decidable. Therefore equivalence is undecidable. □

6 Minimization

We look next at minimization of RNNs. For comparison, state-minimization of a deterministic PFSA is $O(|E| \log |Q|)$ where $|E|$ is the number of transitions and $|Q|$ is the number of states (Aho et al., 1974). Minimization of a non-deterministic PFSA is PSPACE-complete (Jiang and Ravikumar, 1993).

We focus on minimizing the number of hidden neurons ($|N|$) in RNNs:

Minimization: Given RNN R and non-negative integer n, return "yes" if $\exists$ RNN R' with number of hidden units $|N'| \leq n$ such that $R(s) = R'(s)$ for all $s \in \Sigma^*$, and "no" otherwise.

Theorem 13. *RNN minimization is undecidable.*

Proof. We reduce from the Halting Problem. Suppose Turing Machine M decides the minimization problem. For any Turing Machine M', construct the same RNN R as in Theorem 12. We run M on input $\langle R, 0 \rangle$. Note that an RNN with no hidden unit can only output constant $E'_{s,t}$ for all t. Therefore the number of hidden units in R can be minimized to 0 if and only if it always outputs $E'_{s,t}(a) = E'_{s,t}(\$) = 1/2$. If M returns "yes", M' does not halt on ϵ, else it halts. □

7 Conclusion

We proved the following hardness results regarding RNN as a recognizer of weighted languages:

1. Consistency:
   (a) Inconsistent RNNs exist.
   (b) Consistency of RNNs is undecidable.
2. Highest-weighted string:
   (a) Finding the highest-weighted string for an arbitrary RNN is undecidable.
   (b) Finding the highest-weighted string for a consistent RNN is decidable, but the solution length can surpass all computable bounds.
   (c) Restricting to solutions of polynomial length, finding the highest-weighted string is NP-complete.
3. Testing equivalence of RNNs and minimizing the number of neurons in an RNN are both undecidable.

Although our undecidability results are upshots of the Turing-completeness of RNN (Siegelmann and Sontag, 1995), our NP-completeness result is original, and surprising, since the analogous hardness results in PFSA relies on the fact that there are multiple derivations for a single string (Casacuberta and de la Higuera, 2000). The fact that these results hold for the relatively simple RNNs we used in this paper suggests that the case would be the same for more complicated models used in NLP, such as long short term memory networks (LSTMs; Hochreiter and Schmidhuber 1997).

Our results show the non-existence of (efficient) algorithms for interesting problems that researchers using RNN in natural language processing tasks may have hoped to find. On the other hand, the non-existence of such efficient or exact algorithms gives evidence for the necessity of approximation, greedy or heuristic algorithms to solve those problems in practice. In particular, since finding the highest-weighted string in RNN is the same as finding the most-likely translation in a sequence-to-sequence RNN decoder, our NP-completeness result provides some justification for employing greedy and beam search algorithms in practice.

Acknowledgments

This work was supported by DARPA (W911NF-15-1-0543 and HR0011-15-C-0115). Andreas Maletti was financially supported by DFG Graduiertenkolleg 1763 (QuantLA).

References

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. 1974. *The design and analysis of computer algorithms*. Addison-Wesley.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. *OpenFst: A General and Efficient Weighted Finite-State Transducer Library*, Springer Berlin Heidelberg, Berlin, Heidelberg, pages 11–23.

D. Bahdanau, K. Cho, and Y. Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*.

T. L. Booth and R. A. Thompson. 1973. Applying probability measures to abstract languages. *IEEE Transactions on Computers* C-22(5):442–450. https://doi.org/10.1109/t-c.1973.223746.

Francisco Casacuberta and Colin de la Higuera. 2000. Computational complexity of problems on probabilistic grammars and transducers. *Grammatical Inference: Algorithms and Applications Lecture Notes in Computer Science* pages 15–24. https://doi.org/10.1007/978-3-540-45257-7_2.

Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. 2007. L_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science* 18(04):761–779. https://doi.org/10.1142/s0129054107004966.

Manfred Droste, Werner Kuich, and Heiko Vogler. 2013. *Handbook of Weighted Automata*. Springer Berlin.

Vojtch Forejt, Petr Janar, Stefan Kiefer, and James Worrell. 2014. Language equivalence of probabilistic pushdown automata. *Information and Computation* 237:1–11. https://doi.org/10.1016/j.ic.2014.04.003.

T. V. Griffiths. 1968. The unsolvability of the equivalence problem for Λ-free nondeterministic generalized machines. *Journal of the ACM* 15(3):409–413. https://doi.org/10.1145/321466.321473.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Tao Jiang and B. Ravikumar. 1993. Minimal NFA problems are hard. *SIAM Journal on Computing* 22(6):1117–1141. https://doi.org/10.1137/0222067.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. https://arxiv.org/pdf/1602.02410.pdf.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 1317–1327. https://aclweb.org/anthology/D16-1139.

Donald E. Knuth. 1977. A generalization of Dijkstra's algorithm. *Information Processing Letters* 6(1):1–5. https://doi.org/10.1016/0020-0190(77)90002-3.

T. Mikolov and G. Zweig. 2012. Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*. pages 234–239. https://doi.org/10.1109/SLT.2012.6424228.

Mark-Jan Nederhof and Giorgio Satta. 2006. Estimation of consistent probabilistic context-free grammars. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. Association for Computational Linguistics, New York City, USA, pages 343–350. http://www.aclweb.org/anthology/N/N06/N06-1044.

Tibor Radó. 1962. On non-computable functions. *Bell System Technical Journal* 41:877–884.

Géraud Sénizergues. 1997. The equivalence problem for deterministic pushdown automata is decidable. In *Proc. Automata, Languages and Programming: 24th International Colloquium*, Springer Berlin Heidelberg, pages 671–681.

Hava T. Siegelmann and Eduardo D. Sontag. 1995. On the computational power of neural nets. *Journal of Computer and System Sciences* 50(1):132–150. https://doi.org/10.1006/jcss.1995.1013.

Khalil Simaan. 1996. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proc. COLING*. pages 1175–1180. https://doi.org/10.3115/993268.993392.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., pages 3104–3112.

Appendix

Lemma 14. *For every* $k \in \mathbb{N}_+$

$$\prod_{t\in\mathbb{N}} \frac{e^{k(t+1)}}{e^{k(t+1)}+1} = \frac{2}{(-1;\, e^{-k})_\infty} \ ,$$

where $(-1; e^{-k})_\infty$ *is the infinite* e^{-k}*-Pochhammer symbol.*

Proof.

$$\prod_{t\in\mathbb{N}} \frac{e^{k(t+1)}}{e^{k(t+1)}+1} = \prod_{t\in\mathbb{N}_+} \left(\frac{e^{kt}}{e^{kt}+1}\cdot\frac{e^{-kt}}{e^{-kt}}\right)$$

$$= \prod_{t\in\mathbb{N}_+} \frac{1}{1+e^{-kt}} = \left(\left(\prod_{t\in\mathbb{N}_+} \frac{1}{1+e^{-kt}}\right)^{-1}\right)^{-1}$$

$$= \Big(\prod_{t\in\mathbb{N}_+} (1+e^{-kt}) \Big)^{-1} = \Big(\frac{1}{2} \prod_{t\in\mathbb{N}} (1+e^{-kt}) \Big)^{-1}$$
$$= \frac{2}{(-1;\, e^{-k})_\infty} \qquad \square$$

Lemma 15. *Reconsider the RNN of Example 3 and suppose that $h_{s,t}(n) - h_{s,t}(n') = 0$ for all $t \in \mathbb{N}$. Then*

$$\sum_{s\in\Sigma^*} R(s) = 1 - \frac{2}{(-1;\, e^{-2})_\infty} \approx 0.14$$

Proof.

$$\sum_{s\in\Sigma^*} R(s) = \sum_{n\in\mathbb{N}} R(a^n)$$
$$= \sum_{n\in\mathbb{N}} \Big(\frac{e^{-(n+1)}}{e^{n+1}+e^{-(n+1)}} \cdot \prod_{t=0}^{n-1} \frac{e^{t+1}}{e^{t+1}+e^{-(t+1)}} \Big)$$
$$= 1 - \prod_{t\in\mathbb{N}} \frac{e^{2(t+1)}}{e^{2(t+1)}+1} = 1 - \frac{2}{(-1;\, e^{-2})_\infty}$$
$$\approx 0.14 \ ,$$

where the final equality utilizes Lemma 14. $\square$

Lemma 16. *Reconsider the RNN of Example 3 and suppose that there exists a time point $T \in \mathbb{N}$ such that for all $t \in \mathbb{N}$*

$$h_{s,t}(n) - h_{s,t}(n') = \begin{cases} 1 & \text{if } t = T \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$\sum_{s\in\Sigma^*} R(s) = 1$$

Proof.

$$\sum_{s\in\Sigma^*} R(s) = \sum_{n\in\mathbb{N}} R(a^n)$$
$$= \Big(\sum_{n=0}^{T} R(a^n) \Big) + \Big(\sum_{n=T+1}^{\infty} R(a^n) \Big)$$
$$= \sum_{n=0}^{T} \Big(\frac{e^{-(n+1)}}{e^{n+1}+e^{-(n+1)}} \cdot \prod_{t=0}^{n-1} \frac{e^{t+1}}{e^{t+1}+e^{-(t+1)}} \Big)$$
$$+ \sum_{n=T+1}^{\infty} \frac{e^{2n-3T-4}}{e^{n+1}+e^{2n-3T-4}}$$
$$\cdot \Big(\prod_{t=0}^{T} \frac{e^{t+1}}{e^{t+1}+e^{-(t+1)}} \Big) \cdot \Big(\prod_{t=T+1}^{n-1} \frac{e^{t+1}}{e^{t+1}+e^{2t-3T-4}} \Big)$$
$$= \sum_{n=0}^{T} \Big(\frac{1}{e^{2(n+1)}+1} \cdot \prod_{t=0}^{n-1} \frac{e^{2(t+1)}}{e^{2(t+1)}+1} \Big)$$
$$+ \sum_{n=T+1}^{\infty} \frac{e^{n-3T-5}}{1+e^{n-3T-5}} \cdot \Big(\prod_{t=0}^{T} \frac{e^{2(t+1)}}{e^{2(t+1)}+1} \Big)$$
$$\cdot \Big(\prod_{t=T+1}^{n-1} \frac{1}{1+e^{t-3T-5}} \Big)$$
$$= 1 - \Big(\prod_{t=0}^{T} \frac{e^{2(t+1)}}{e^{2(t+1)}+1} \Big) + \Big(\prod_{t=0}^{T} \frac{e^{2(t+1)}}{e^{2(t+1)}+1} \Big)$$
$$\cdot \sum_{n=T+1}^{\infty} \frac{e^{n-3T-5}}{1+e^{n-3T-5}} \cdot \Big(\prod_{t=T+1}^{n-1} \frac{1}{1+e^{t-3T-5}} \Big)$$
$$= 1 - \Big(\prod_{t=0}^{T} \frac{e^{2(t+1)}}{e^{2(t+1)}+1} \Big)$$
$$\cdot \left(1 - 1 + \prod_{t=T+1}^{\infty} \frac{1}{1+e^{t-3T-5}} \right)$$
$$= 1 - \Big(\prod_{t=0}^{T} \frac{e^{2(t+1)}}{e^{2(t+1)}+1} \Big) \cdot \Big(\prod_{t=T+1}^{\infty} \frac{1}{1+e^{t-3T-5}} \Big)$$
$$\geq 1 - \Big(\prod_{t=0}^{T} \frac{e^{2(t+1)}}{e^{2(t+1)}+1} \Big) \cdot \Big(\prod_{t\in\mathbb{N}} \frac{1}{1+e^{t}} \Big)$$
$$= 1 \qquad \square$$

Lemma 17.

Proof. We set $h_{s,-1}(n) = h_{-1}(n)$ for all $n \in N$ and $h'_{s,-1}(n') = h'_{-1}(n')$ for all $n' \in N'$. Then trivially $h'_{s,-1}(n_1) - h'_{s,-1}(n_2) = h_{-1}(n) - 0 = h_{s,-1}(n)$. Moreover, $h'_{s,t}(n_1) = \sigma\langle V \cdot h'_{s,t-1} + V'_{s[t]} \rangle(n_1)$

$$= \sigma\langle \sum_{n'\in N'} V(n_1,n') \cdot h'_{s,t-1}(n')$$
$$+ V'_{s[t]}(n_1) \rangle$$
$$= \sigma\langle \sum_{n'\in N} \big(V(n_1,n'_1) \cdot h'_{s,t-1}(n'_1)$$
$$+ V(n_1,n'_2) \cdot h'_{s,t-1}(n'_2) \big) + V'_{s[t]}(n_1) \rangle$$
$$= \sigma\langle \sum_{n'\in N} U(n,n') \cdot \big(h'_{s,t-1}(n'_1) - h'_{s,t-1}(n'_2) \big)$$
$$+ U'_{s[t]}(n) \rangle$$
$$= \sigma\langle \sum_{n'\in N} U(n,n') \cdot h_{s,t-1}(n') + U'_{s[t]}(n) \rangle$$

Similarly, we can show that $h'_{s,t}(n_2) =$

$$\sigma\langle \sum_{n'\in N} U(n,n') \cdot h_{s,t-1}(n') + U'_{s[t]}(n) - 1 \rangle$$

Hence $h'_{s,t}(n_1) - h'_{s,t}(n_2) = h_{s,t}(n)$ as required. $\square$